# PATENT ABSTRACTS OF JAPAN

(11)Publication number : 06-202882
(43)Date of publication of application : 22.07.1994

(51)Int.Cl. G06F 9/46

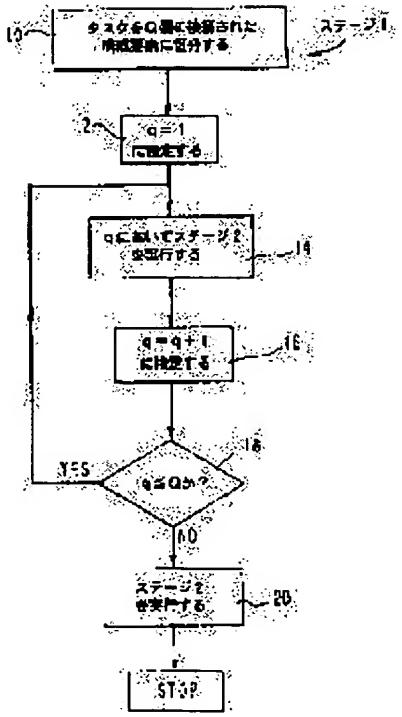
(21)Application number : 05-238548
(22)Date of filing : 24.09.1993

(71)Applicant : INTERNATL BUSINESS MACH CORP <IBM>
(72)Inventor : CHEN MING-SYAN
TUREK JOHN J E
WOLF JOEL L
YU PHILIP SHI-LUNG

(30)Priority
Priority number : 92 968717 Priority date : 30.10.1992 Priority country : US

## (54) SCHEDULING METHOD FOR A PLURALITY OF TASKS AND JOBS

(57)Abstract:
PURPOSE: To provide an efficient and effective solution method for the flexible scheduling problems of tasks having priority.
CONSTITUTION: This scheduling method for a plurality of tasks and jobs consists of stages 1 to 3. The stage 1 is preliminary and divides the tasks into connected components (10). A prescribed job is calculated (14) at the stage 2 and each schedule of tasks contained in the job is decided according to its priority constraint. The stage 2 is repeated to calculate the execution time of jobs. The schedule of jobs having no priority constraint are decided (20) at the stage 3.



(19)日本国特許庁（ＪＰ）　(12)**公開特許公報（Ａ）**　(11)特許出願公開番号

特開平6－202882

(43)公開日　平成6年(1994)7月22日

| (51)Int.Cl.[5] | 識別記号 | 庁内整理番号 | ＦＩ | 技術表示箇所 |
|---|---|---|---|---|
| Ｇ０６Ｆ　9/46 | ３４０　Ｂ | 8120－5Ｂ | | |

審査請求　有　　請求項の数7　（全 13 頁）

(21)出願番号　特願平5－238548

(22)出願日　平成5年(1993)9月24日

(31)優先権主張番号　９６８７１７

(32)優先日　1992年10月30日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　390009531
インターナショナル・ビジネス・マシーンズ・コーポレイション
INTERNATIONAL BUSINESS MASCHINES CORPORATION
アメリカ合衆国10504、ニューヨーク州アーモンク　（番地なし）

(72)発明者　ミンーシャン　チェン
アメリカ合衆国10598、ニューヨーク州ヨークタウン　ハイツ、ブレンダー　レイン　710

(74)代理人　弁理士　合田　潔　（外６名）

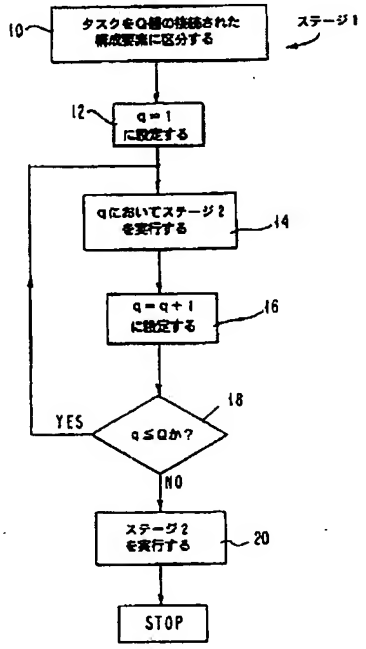
最終頁に続く

(54)【発明の名称】　複数のタスク及びジョブのスケジューリング方法

(57)【要約】

【目的】　優先順位を備えたタスクにおいて柔軟なスケジューリングの問題の効率的かつ効果的な解決法を提供する。

【構成】　複数のタスク及びジョブのスケジュール方法は、ステージ１、２及び３から成る。ステージ１は予備ステージで、接続された構成要素にタスクを区分する(10)。ステージ２で、所定のジョブの計算を行い(14)、ジョブ内のタスクの各々のスケジュールを優先順位制約に従って決定する。このステージを繰り返してジョブ実行時間の計算を行う。ステージ３では、優先順位制約のないジョブのスケジュールを決定する(20)。



1

【特許請求の範囲】

【請求項１】　並列に作動する複数のプロセッサに優先順位制約を有する複数のタスクをスケジュールするための方法であって、

（ａ）複数のジョブを定義するステップを備え、前記ジョブの各々が前記タスクの一部と単一のジョブ内に含まれる前記タスクの対のみに関連する全ての優先順位の制約を含み、

（ｂ）各前記ジョブにおいて、前記ジョブの前記タスクに複数のタスクスケジュールを作成するステップを備え、前記タスクスケジュールの各々が前記ジョブに割り当てられる可能性のある異なる数の前記プロセッサに対応し、前記ジョブの前記タスク間でのあらゆる優先順位制約を尊重し、

（ｃ）前記タスクスケジュールの各々に推定されたジョブ実行時間を決定するステップと、

（ｄ）前記ジョブの各々及び前記ジョブの各々に割り当てられる各異なる数のプロセッサに前記推定されたジョブ実行時間を用いて、前記ジョブの各々へのプロセッサの割り当てを決定するステップと、

（ｅ）前記決定された割り当てを用いて前記ジョブにジョブスケジュールを作成するステップと、

（ｆ）ステップ（ｅ）で作成されたジョブスケジュールを用いて前記プロセッサに前記ジョブを実行するステップと、

から成る複数のタスクのスケジューリング方法。

【請求項２】　ステップ（ａ）で定義された前記ジョブの各々がデータベース照会として定義される、請求項１に記載の複数のタスクのスケジューリング方法。

【請求項３】　ステップ（ｂ）が前記ジョブの前記タスクの優先順位制約を尊重するように前記ジョブ内の前記タスクを順序付けるステップを含む、請求項１に記載の複数のタスクのスケジューリング方法。

【請求項４】　ステップ（ｄ）が、前記タスクスケジュールの各々毎に推定されたジョブ実行時間とそれに対応するプロセッサの数の積を計算するステップを含み、前記ジョブの各々へのプロセッサの最初の割り当てが前記ジョブの各々にそのように計算された積の内の最小の積に対応する、請求項１に記載の複数のタスクのスケジューリング方法。

【請求項５】　ステップ（ｅ）で作成された前記ジョブスケジュールが２次元ビンパッキング方法を用いて作成される、請求項１に記載の複数のタスクのスケジューリング方法。

【請求項６】　前記ジョブスケジュールの前記ジョブのいずれかの内に遊休プロセッサの時間があるかどうかについてステップ（ｅ）で作成された前記ジョブスケジュールを解析し、そのような遊休プロセッサの時間があるならば、ステップ（ｄ）において決定された前記割り当てを改良し、前記改良された割り当てに基づいて新たなジョブスケジュールを作成するステップと、を更に含む請求項１に記載の複数のタスクのスケジューリング方法。

2

【請求項７】　並列に作動する複数のプロセッサを有する計算システムで複数の別個のジョブをスケジュールするための方法であって、前記ジョブの各々が少なくとも１つのタスクから成り、少なくとも幾つかの前記ジョブが優先順位制約のありうる複数のタスクから成り、前記プロセッサの各々が別個のタスクを同時に実行することが可能であり、

（ａ）各ジョブにおいて推定されたジョブ実行時間のセットを作成するステップを備え、あらゆる前記セットの各前記推定されたジョブ実行時間が前記各ジョブの実行に専用可能な異なる数の前記プロセッサに対応し、前記推定されたジョブ実行時間が生成されると共に、前記各ジョブから成るタスクのあらゆる優先順位制約に従い、

（ｂ）各前記推定されたジョブ実行時間において、ジョブ実行時間と対応するプロセッサの数の積を計算するステップと、

（ｃ）推定されたジョブ実行時間の各前記セットにおいて、前記計算された積の内の最小の積を識別し、前記各ジョブの実行のために前記識別された最小の積に対応する前記プロセッサの数を試験的に割り当てるステップと、

（ｄ）各前記ジョブに前記試験的に割り当てられた数のプロセッサを用い、前記ジョブの各々の内の前記タスクに２次元ビンパッキングアルゴリズムを使用することによって、並列にある前記ジョブの全てを実行するための試験的な全体スケジュールを作成するステップと、

（ｅ）前記試験的な全体スケジュールに対応する全体実行時間を推定し、前記推定された全体実行時間を記録するステップと、

（ｆ）前記試験的な全体スケジュールに遊休プロセッサがあるかどうかを決定するステップと、

（ｇ）遊休プロセッサがあるならば、識別されたジョブを実行するために試験的に割り当てられたプロセッサの最大数よりも少なく、遊休プロセッサの時間の最大量と対応する前記全体スケジュールにおいてジョブを識別するステップと、

（ｈ）前記識別されたジョブを実行するために試験的に割り当てられたプロセッサの数を増加するステップと、

（ｉ）前記試験的な全体スケジュールの前記遊休プロセッサの時間が、増加によってもはや減少できなくなるまでステップ（ｄ）乃至（ｆ）を繰り返すステップと、

（ｊ）前記記録された全体の実行時間から最小の全体実行時間を選択し、前記プロセッサで並列にある前記ジョブを実行するために前記最小の記録された全体実行時間に対応する前記試験的な全体スケジュールを用いるステップと、

から成る複数のジョブのスケジューリング方法。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、概して最後のタスクが終了する終了時間を最小限にするように並列処理コンピュータシステムで実行するための複数のタスクをスケジュールすることに関し、特に、タスク間に優先順位制約があるようなタスクをスケジュールすると共に、同時にスケジュール内の当該優先順位制約を尊重することに関する。例えば、タスクは複数のデータベース照会内で個々のステップであってもよい。

【０００２】

【従来の技術】データベースからの情報要求（一般的に「データベース照会(database queries)」と称される）は概して数多くのステップ（本明細書では「タスク(task)」と称される）を含み、照会における少なくとも１つのタスクの結果（本明細書では「ジョブ(job)」と称される）は同一のジョブにおける後のタスクで用いられる。これは、ジョブ（もう１つのタスクで使用される結果を生じる）内のいかなる先のタスク（単数又は複数）も、当該結果を用いるいかなる後のタスクの実行が開始する前に終了されなければならない（即ち、「優先(precede)」しなければならない）ために、優先順位制約と称される。

【０００３】並列処理システムは複数の処理要素を提供し、処理要素の各々は、他の処理要素が同一タスク若しくはジョブ又は異なるタスク若しくはジョブの他のタスク又は他のタスク部分を処理するのと同時に、ジョブの全体のタスク又はそのようなタスクの一部を実行することができる。複数のジョブが並列処理システムで同時に実行されるとスケジュールされるとき、最小限の時間で全てのジョブにおいてタスクの総数の最後のタスクを終了するスケジュールを作成すると共に、勿論、タスク間の全ての優先順位制約を尊重することが一般的な目的となる。

【０００４】本発明の目的のために、並列処理システムにおける各処理要素（以下本明細書で処理要素というよりもむしろ「プロセッサ(processor)」と称される）は、（単独又は少なくとも１つの他のプロセッサとの組合せで）一度に単一のタスクしか処理せず、もう１つのタスクを開始するために使用可能になる前に（単独又は少なくとも１つの他のプロセッサとの組合せで）タスクを処理して終了する。並列処理システムの物理資源が同時に１より多くのタスクを実行できるならば、プロセッサが１より多いと見られるだけである。

【０００５】各ジョブはまた、割り当てられたプロセッサの数が非増加関数である実行時間を有すると仮定される。ジョブの実行時間は、ジョブの実行に割り当てられたプロセッサの数の増加と共に増加できない。これは、追加のプロセッサの使用(USE)によってジョブの終了時間が長くなる場合、最良のスケジュールはその追加されたプロセッサを遊休(IDLE)にすることであり、従って、少なくともジョブの実行時間はもう１つのプロセッサの追加によって増加しないからである。

【０００６】問題は、各ジョブへのプロセッサの割り当て（即ち、使用するプロセッサの数）、及び、全体として、最後のタスクが終了する終了時間を最小限にするように、全てのジョブのタスクをいかなる優先順位制約も尊重するプロセッサへ割り当てるスケジュールを見つけることである。

【０００７】従って、Ｐ個のプロセッサから成るマルチプロセッサコンピュータシステム、並びに、このシステムでスケジュールされるＮ個のタスクのセットを考える。タスクは、優先順位関係

【０００８】

【外１】

$$\prec$$

【０００９】によって部分的に順序付けられる。各タスク$j \in \{1,\ldots,N\}$が任意の数のプロセッサ$\beta_i \in \{1,\ldots,P\}$に割り当てられ、タスク実行時間$t_i(\beta_i)>0$が割り当てられたプロセッサの数の非増加関数であると仮定する。タスクに割り当てられた全てのプロセッサは、そのタスクを一致して実行することが必要とされる。すなわち、これらのプロセッサ$\beta_i$は全て任意の開始時間、例えば、$\tau_i$、でタスクｊを開始することが必要とされる。次にプロセッサは任意の後の終了時間$\tau_i + t_i(\beta_i)$でタスクｊを終了する。各タスク$j \in \{1,\ldots,N\}$において、スケジュールはプロセッサ割り当て$\beta_i$及び開始時間$\tau_i$から構成される。スケジュールは、以下の２つの意味で正当であることが必要とされる。・任意の時間τにおいて、アクティブプロセッサの数はプロセッサの総数を越えない。換言すれば次の通りである。

【００１０】

【数１】

$$\sum_{\{j \mid \tau_j \le \tau < \tau_j + t_j(\beta_j)\}} \beta_j \le P$$

【００１１】・１つのタスク$j_1$がもう１つのタスク$j_2$に優先するならば、第２タスクは第１タスクが終了するまで開始できない。換言すれば次の通りである。

【００１２】

【外２】

$$j_1 \prec j_2$$

【００１３】ならば、$\tau_{i1} + t_{i1}(\beta_{i1}) \le \tau_{i2}$である。最適なスケジュールを見つけることが問題であり、

【００１４】

【数２】

$$\max_{1 \le j \le N} \{\tau_j + t_j(\beta_j)\}$$

【００１５】によって与えられる全体のメイクスパン(makespan)が最小限にされる。換言すれば、最後のタスク終了時間を最小限にすることが目的である。これは柔軟な(malleable) スケジューリングの問題と称される。

【００１６】図１は、３つのジョブに対する柔軟なスケジューリング問題を示すブロック図である。入力はグラフのノード（節）によって表される３つのジョブから成るタスクと、エッジ（即ち、相互接続ライン）によって表される優先順位関係とから構成される。２つのノードがエッジによって接続されるならば、下位ノードに対応するタスクは上位ノードに対応するタスクが開始する前に終了しなければならない。スケジューリングプロシージャ（手順）は、出力として優先順位関係を尊重するタスクのスケジュールを作成する。プロセッサは横軸に、時間は縦軸に示されている。スケジュールのメイクスパンもまた示されている。

【００１７】ドュー(J. Du) 及びロイング(J. Leung) 著、「並列タスクシステムのスケジューリングの複雑さ(Complexity of Scheduling Parallel Task Systems)」(SIAMJournal of Discrete Mathematics (1989)は、柔軟なスケジューリングの問題は、優先順位制約がない特別の場合でさえも強い意味においてNP-hard であることを示している。（数学的に、これはその問題への全体的に最適な解決法を見つけるための効率的なプロシージャが発見される可能性は極めて低いことを意味する。）解決を試みるいかなるプロシージャも、この文献には展開されていない。

【００１８】チェン(M.-S. Chen)、ユー(P. Yu) 及びウー(K.-L. Wu)著、「マルチプロセッサシステムにおける複数の連結照会を実行するためのスケジューリング及びプロセッサの割り当て(Scheduling and Processor Allocation for the Esecution of Multi- Join Queries in a Multiprocessor System) 」(Proc. IEEE Conference on Data Engineering (1992) では、優先順位及び一般的なタスク実行時間を備えた単一の照会のコンテキストにおける柔軟なスケジューリングの問題が考慮されている。この文献では複数の照会の場合が考慮されておらず、単一の照会のために開発されたプロシージャは必ずしもあまり効果的ではない解決法を見つけている。

【００１９】柔軟なスケジューリングの問題の非優先順位バージョンが、先の３つの文献で研究されている。これら文献の各々は、一般的なタスク実行時間を考慮し、少なくとも１つのプロシージャを提案していた。

【００２０】テューレック(J. Turek)、ウルフ(J. Wolf) 及びユー(P. Yu) 著、「並列可能なタスクをスケジュールするための近似アルゴリズム(Approximate Algorithmsfor Scheduling Parallelizable Tasks)」(Proc. Symposium on Parallel Algorithms and Architectures (1992)では、柔軟なプロシージャのクラスが提示され、プロシージャの各々が優先順位のない非柔軟なスケジューリングのためのプロシージャに基づき、プロシージャの各々が対応するより一層単純なプロシージャの最悪の場合の漸近の実行と一致する。しかしながら、優先順位はこの文献では考慮されていない。

【００２１】テューレック(J. Turek)、ウルフ(J. Wolf) 、パティパティ(K. Pattipati)及びユー(P. Yu) 著、「並列可能なタスクのスケジューリング: 全てをシェルフに置く(Scheduling Parallelizable Tasks: Putting it all on the Shelf)」(Proc. ACM Sigmetrics Conference (1992) では、全ての可能なシェルフに基づく解決法のセットに対して最適なシェルフプロシージャが開発されている。（シェルフ解決法は、スケジューリング問題を解決するための方法のクラスを表す。）優先順位はこの文献でもまた考慮されていない。

【００２２】クリシュナムルティ(R. Krishnamurthi)及びマー(E. Ma) 著、「区分可能なマルチプロセッサシステムにおいて変化する区分サイズのタスクをスケジュールするための近似アルゴリズム(An Approximation Algorithm for Scheduling Taskson Varying Partition Sizes in Partitionable Multiprocessor Systems)」(IBM Research Report 15900 (1990) では、単一のシェルフに並列可能なタスクをパックする特別な場合を解決している。従って、この文献にも優先順位は考慮されていない。

【００２３】

【発明が解決しようとする課題】従って、本発明の目的は、優先順位を備えたタスクにおいて柔軟なスケジューリングの問題の効率的かつ効果的な解決法を提供することである。

【００２４】本発明のもう１つの目的は、各ジョブが優先順位制約の可能性があるタスクの内の少なくとも１つのタスクから成る、複数のジョブを並列処理システムにスケジュールするための方法を提供することである。

【００２５】本発明の目的はまた、並列処理システムの複数のデータベース照会を効率的にスケジュールするための方法を提供することである。

【００２６】

【課題を解決するための手段と作用】これらの目的及び利点並びに追加の目的及び利点は、本発明によって達成される。本発明は、次の２つの内の第１番目のステージのみが優先順位制約を含む、スケジューリングの２つの別個のステージ（本明細書では、プロシージャの第２ステージ及び第３ステージと称される）を用いることによって、柔軟なスケジューリングの問題に対する効果的な解決法を決定するための効率的なプロシージャを提供する。

【００２７】プロシージャの第１ステージ（予備でありしばしばオプションである）では、スケジュールされるべきタスクの全体のセットはジョブに区分され、各優先順位制約は単一のジョブにしか影響を及ぼさない。（即ち、特定のタスクが入力のために依存する全てのタスクは、その特定のタスクとして同一のジョブ内にも含まれる。）多くの場合、この条件を満たす適切なジョブは既に存在し、この第１ステージにおけるあらゆる実際の計算を不要にすることもある。これは、例えば、各照会が通常既に本質的にこの条件を満たしているデータベース照会の場合に一般的である。そのような場合には、各照会を別個のジョブとして定義するのが必要なだけである。

【００２８】プロシージャの第２ステージ（実際のスケジューリングの第１ステージ）では、各ジョブは何度も別個で最適にスケジュールされ（そのジョブの実行に割り当てられる可能性のある異なる数の各プロセッサに一度）、各ジョブへのこれら他の最適なスケジュールの各々（本明細書では時々タスクスケジュールと称される）が作成されると共に、ジョブのタスク間での優先順位制約を尊重する。各ジョブにおけるこれら他の最適なスケジュールの各々の推定されたジョブ実行時間がまた決定され、このプロシージャの第３ステージで使用するために記録される。優先順位を尊重する最適なスケジュールを作成するためのあらゆる公知の方法が、この第２ステージで用いられる。「ダイナミックプログラミング(dynamic programming)」を用いて、この作成を行うための好ましい方法が開示されている。

【００２９】プロシージャの第３ステージ（実際のスケジューリングの第２ステージ）では、このプロシージャの第２ステージで作成される各ジョブへの他の推定されたジョブ実行時間のセットは、入力として用いられて全てのジョブに最適な全体スケジュールを作成する。優先順位制約の各々が第２ステージ（即ち、第１スケジューリング段階）で既に完全に尊重されているため、このプロシージャの第３ステージは第２ステージで作成された推定されたジョブ実行時間からこの最適な全体スケジュールを作成するためにいかなる公知の方法を用いてもよい。ジョブのみがこのステージでスケジュールされるために、プロシージャの第３ステージでは尊重すべき優先順位制約はない。かつ、このプロシージャの第１ステージにおいて、ジョブ同士の間に優先順位制約がないようにジョブが定義されるために、ジョブ同士の間に優先順位制約はない。これは、この第３ステージであらゆるスケジューリング方法の使用も可能にするために故意に行われる。

【００３０】このプロシージャの第３ステージにおいて、プロシージャの第２ステージで作成された推定されたジョブ実行時間の各セットの特定のメンバ（各ジョブに１セット）は、その選択に対応するプロセッサの数と共に選択され、ジョブの全体スケジュールはこれらの選択を用いて作成される。概して、ジョブに対するプロセッサを割り当てるための多くの異なる選択も同様にスケジュールされる。全てのジョブの最小限の終了時間を有する全体スケジュールは、最適な全体スケジュールとして選択される。この選択を行う好ましい方法が記載されている。

【００３１】記載された第３ステージの好ましい方法において、推定されたジョブ実行時間の各セットから、セットの効率的なメンバが選択され（即ち、対応する推定されたジョブ実行時間に関連して使用されるプロセッサの数が、そのタスクの実行に割り当てられたプロセッサの効率的な使用を反映するような意味において効率的に見えるもの、並びに、推定されたジョブ実行時間と使用されたプロセッサの数の積が最も小さいものが好ましい）、このメンバに対応するプロセッサの数が、そのジョブに割り当てられたプロセッサの最初の数として試験的に選択される。多くのプロセッサを各ジョブに対して割り当てるためのこの最初の試験的な選択を用いることによって、全体スケジュールはあらゆる公知の２次元ビンパッキング（詰め）アルゴリズムを用いて作成される。プロセッサを割り当てるためのこの最初のセットへの全てのジョブの推定された全体の実行時間が決定及び記録され、全体スケジュールで浪費された（即ち、遊休の）プロセッサの時間があるかどうかについて解析される。浪費されたプロセッサの時間があるならば、最も浪費されたプロセッサの時間と対応するジョブが識別され、そのジョブに割り当てられたプロセッサの数が１だけ増加され、新たな全体スケジュールが作成される。以前に記録された最良の時間より良ければ、この新たな全体スケジュールの全体の実行時間もまた推定及び記録される。新たな全体スケジュールもまた、浪費されたプロセッサの時間があるかどうかについて解析される。このプロセスは、あらゆる浪費されたプロセッサの時間がなくなるまで続き、最良の（即ち、最小の）全体実行時間を作成した全体スケジュールが最適なスケジュールとして選択されて、ジョブを実行するのに使用される。

【００３２】第２ステージの好ましい方法において、ジョブのタスクはまずトポロジカル（構成）順に分類されるが、これはジョブのタスク間の優先順位制約の全てを尊重する順序である。トポロジカル順序は枝が相互に連結したタスクの木として表され、葉はジョブのタスクとして、枝の相互連結はタスク間の優先順位制約として表される。

【００３３】次に、葉から始めて木の幹に向かって戻ると、各タスク（及びそのタスクの全ての子タスク）は何度も最適にスケジュールされ（即ち、そのタスク及び、もしあれば、その子タスクを実行するために使用されてもよい各可能な数のプロセッサにつき一度）、そのタスクに可能な他の最適なスケジュールのセット（そのタス

クに割り当てられてもよい各可能な数のプロセッサにつき１つの他の最適なスケジュール）を作成する。タスクが何度も交互に最適にスケジュールされつつあり（各異なるプロセッサの割り当ての可能性につき一度）、タスクが少なくとも１つの子タスクを有するときは常に、子タスクもまた親タスクと共に最適にスケジュールされる。これは優先順位制約を尊重するために実行される。このスケジュールを効率的に実行するために、最適なスケジュールと、子タスクがプロシージャの初期において個々に何度も交互に最適にスケジュールされたときに作成された子タスクの推定された実行時間とのセットが使用される。

【００３４】タスクが直列と同様に並列で実行可能な２つの子タスクを有するならば（２つの子タスクの間に優先順位制約がないため）、各プロセッサに割り当てられる数の最適なスケジュールは、１つの可能性として互いに並列で実行され、他の可能性として互いに直列で実行され、勿論、互いに並列でスケジュールされるときには各子タスクに対するプロセッサの各可能な割り当てを分割して、それら子タスクが他のスケジュールを作成することによって決定される。

【００３５】各タスクに関して、最適なスケジュールはそのタスクに割り当てられてもよい各可能な数のプロセッサで作成され、（そのような各最適なスケジュールの推定された実行時間と共に）これら最適なスケジュールが記録される。先に述べたように、子タスクの最適なスケジュール及び推定された実行時間のセットはその子タスクの親タスク（単数又は複数）を最適にスケジュールし、ジョブにおける最後の親タスク（即ち、ルートタスク）は、第３ステージのジョブで使用されるセットとなる最適なスケジュール及び対応する推定された実行時間のセットを作成する。

【００３６】並列に作動する複数のプロセッサに優先順位制約を有する複数のタスクをスケジュールするための方法は、（ａ）複数のジョブを定義するステップを備え、前記ジョブの各々が前記タスクの一部と単一のジョブ内に含まれる前記タスクの対のみに関連する全ての優先順位の制約を含み、（ｂ）各前記ジョブにおいて、前記ジョブの前記タスクに複数のタスクスケジュールを作成するステップを備え、前記タスクスケジュールの各々が前記ジョブに割り当てられる可能性のある異なる数の前記プロセッサに対応し、前記ジョブの前記タスク間でのあらゆる優先順位制約を尊重し、（ｃ）前記タスクスケジュールの各々に推定されたジョブ実行時間を決定するステップと、（ｄ）前記ジョブの各々及び前記ジョブの各々に割り当てられる各異なる数のプロセッサに前記推定されたジョブ実行時間を用いて、前記ジョブの各々へのプロセッサの割り当てを決定するステップと、（ｅ）前記決定された割り当てを用いて前記ジョブにジョブスケジュールを作成するステップと、（ｆ）ステップ（ｅ）で作成されたジョブスケジュールを用いて前記プロセッサに前記ジョブを実行するステップと、から成る。

【００３７】並列に作動する複数のプロセッサを有する計算システムで複数の別個のジョブをスケジュールするための方法は、前記ジョブの各々が少なくとも１つのタスクから成り、少なくとも幾つかの前記ジョブが優先順位制約のありうる複数のタスクから成り、前記プロセッサの各々が別個のタスクを同時に実行することが可能であり、（ａ）各ジョブにおいて推定されたジョブ実行時間のセットを作成するステップを備え、あらゆる前記セットの各前記推定されたジョブ実行時間が前記各ジョブの実行に専用可能な異なる数の前記プロセッサに対応し、前記推定されたジョブ実行時間が生成されると共に、前記各ジョブから成るタスクのあらゆる優先順位制約に従い、（ｂ）各前記推定されたジョブ実行時間において、ジョブ実行時間と対応するプロセッサの数の積を計算するステップと、（ｃ）推定されたジョブ実行時間の各前記セットにおいて、前記計算された積の内の最小の積を識別し、前記各ジョブの実行のために前記識別された最小の積に対応する前記プロセッサの数を試験的に割り当てるステップと、（ｄ）各前記ジョブに前記試験的に割り当てられた数のプロセッサを用い、前記ジョブの各々の内の前記タスクに２次元ビンパッキングアルゴリズムを使用することによって、並列にある前記ジョブの全てを実行するための試験的な全体スケジュールを作成するステップと、（ｅ）前記試験的な全体スケジュールに対応する全体実行時間を推定し、前記推定された全体実行時間を記録するステップと、（ｆ）前記試験的な全体スケジュールに遊休プロセッサがあるかどうかを決定するステップと、（ｇ）遊休プロセッサがあるならば、識別されたジョブを実行するために試験的に割り当てられたプロセッサの最大数よりも少なく、遊休プロセッサの時間の最大量と対応する前記全体スケジュールにおいてジョブを識別するステップと、（ｈ）前記識別されたジョブを実行するために試験的に割り当てられたプロセッサの数を増加するステップと、（ｉ）前記試験的な全体スケジュールの前記遊休プロセッサの時間が、増加によってもはや減少できなくなるまでステップ（ｄ）乃至（ｆ）を繰り返すステップと、（ｊ）前記記録された全体の実行時間から最小の全体実行時間を選択し、前記プロセッサで並列にある前記ジョブを実行するために前記最小の記録された全体実行時間に対応する前記試験的な全体スケジュールを用いるステップと、から成る。

【００３８】

【実施例】以下の記載において、簡潔性のために、タスク同士の間の優先順位制約が森を形成すると仮定する。（数学的に、これはエッジのシーケンスを介して所定のノードへ接続される全てのノードのセットが木を形成することを意味する。このセットは、所定のノードの接続

された構成要素と称される。そのような木の集合を森と称する。）この仮定は緩めることができる。図2を特に参照すると、本発明のステージから成るブロック図が示されている。

【0039】ブロック10で示されるステージ1は、本発明の予備ステージである。ステージ1は接続された構成要素にタスクを区分して、iとjの双方が同じ区画にある場合にのみ、

【0040】

【外3】

$$i \prec j$$ である。

【0041】換言すれば、各qは森の木に正確に対応する。Q個のそのような接続された構成要素があると仮定する。区分のためのプロシージャは、コーレム(T. Corem)、リーサーソン(C. Leiserson)、リベスト(R. Rivest)によるアルゴリズム(McGrawHill (1992), 441-442頁)に見られる。タスクのQ個のセットの各々が単一のジョブから構成されるとして考えるのは好都合である。（勿論、ジョブの識別子が予め知られている場合は、そのようなプロシージャは予備段階として必要とされない。）代表となるデータベースの例において、ジョブは個々の照会及び照会内のステップに対するタスクに対応する。照会の識別子は予め知られている可能性が高い。

【0042】本発明の残りは事実上階層的である。2つの追加のステージ、即ち、ステージ2及びステージ3がある。

【0043】1とQの間の所定のジョブqにおけるステージ2の計算がブロック14に示されている。この計算はブロック12で開始され、ブロック16及び18によって制御される。1とプロセッサの総数Pの間にあるプロセッサの各数pにおいて、ステージ2はジョブq内のタスクの各々のスケジュールを決定する。このスケジュールは関連する優先順位制約に従う。このステージを繰り返し用いることによって、各q及び各pのジョブ実行時間の計算が行われる。ステージ2のより詳細な記述は以下に示される。

【0044】ステージ3はブロック20に示されている。このステージは、ジョブ同士の間に優先順位制約のないQ個のジョブでスケジュールを決定する。ステージ3はステージ2の出力を入力として使用する。ステージ3のより詳細な記述は以下に示される。

【0045】ステージ2の詳細は図3のブロック図に示されている。ステージ2のプロシージャの性質を理解するために、図4に示されるジョブ優先順位ツリーを考える。（ノードはタスクを表す。2つのノードがエッジによって接続されるならば、上位ノードに対応するタスクが開始する前に下位ノードに対応するタスクが終了しなければならない。）ステージ2は以下の特性に従うという意味で連続する改良されたプロシージャである。即ち、タスクが所定のプロセッサに割り当てられるならば、全ての子（及び結果として全ての子孫全体）はそれらプロセッサのサブセットに割り当てられる。従って、ステージ2のプロシージャは非常に強い方法で優先順位ツリーの構造を尊重する。

【0046】図5は、図4に示されるジョブの可能なステージ2の解決法を示している。陰影の付いた領域は浪費された作業を表す。横軸はプロセッサを表し、縦軸は時間を表す。なお、ルートタスク（ノード12）は全てのプロセッサに割り当てられる。ノード12の2つの子、即ち、ノード10及び11、はそれ自体がサブツリーのルートである。ノード10に対応するサブツリー全体は、並列として特徴付けられるようにノード11と対応するサブツリー全体の左側にパックされる。左側のサブツリーを降りながら、ノード10の子と対応するツリー、即ち、ノード7及び8のツリーを考える。ノード7と対応するサブツリーは、パッチとして特徴付けられるようにノード8と対応するサブツリーより上にパックされる。実際、ステージ2のプロシージャは、所定のノードの全ての子に対応するサブツリーが並列方法又はパッチ方法でパックされる特性に従う。

【0047】ステージ2のプロシージャは、上記2つの特性を尊重する最適なスケジュールを見つける。図3を再び参照する。特に、本発明のステージ2はタスクをトポロジカルに順序付けることによって開始するため、

【0048】

【外4】

$$j_1 \prec j_2$$

【0049】は$j_1 < j_2$を意味する。これはブロック30に示されている。トポロジカル分類のプロシージャは、コーレム(T. Corem)、リーサーソン(C. Leiserson)、リベスト(R. Rivest)によるアルゴリズム(McGraw Hill (1992), 485-488頁)に見られる。次に、ダイナミックプログラミングを用いて、葉のノードを上へ、より小さな数のプロセッサからより大きな数のプロセッサの順に従って進む。p個のプロセッサを使用するタスクj及びその全ての子孫において、ステージ2のプロシージャによって見つけられる最適なメイクスパンを、

【0050】

【外5】

$$\hat{t}(p)$$

【0051】として定義する（ここで、$1 \leq p \leq P$とする）。

【0052】

【外6】

$$\hat{t}(p)$$

【0053】を生成するため、ステージ2のプロシージャはp個のプロセッサを使用してタスクjをバックし、いずれも良い（a）パッチ方法、又は（b）最も可能な

並列方法でバックされた、ｊの子のサブツリーの全てのバッキングをタスクｊより下に追加する。

【００５４】ブロック３６は、所定のタスクｊ及びプロセッサの数ｐにおける

【００５５】

【外７】

$$\hat{t}_j(p)$$

【００５６】の計算を示す。ｐループの開始がブロック３２に示され、ｐループはブロック４２及び４４によって制御される。ｊループの開始がブロック３４に示され、ｊループはブロック３８及び４０によって制御される。

【００５７】

【外８】

$$\hat{t}_j(p)$$



＊【００５８】を計算するため、$CH_1$がｊの子の数を示すとする。$CH_1 = 0$ならば、

【００５９】

【数３】

$$\hat{t}_j(p) = t_j(p)$$

【００６０】に設定する。$CH_1 = 1$ならば、

【００６１】

【数４】

$$\hat{t}_j(p) = t_j(p) + \hat{t}_{ch_1 j}(p)$$

【００６２】に設定する。$CH_1 = 2$ならば、

【００６３】

【数５】

＊

$$\hat{t}_j(p) = t_j(p) + \min\{\hat{t}_{ch_1 j}(p) + \hat{t}_{ch_2 j}(p),\ \min_{1<q<p} \max\{\hat{t}_{ch_1 j}(q), \hat{t}_{ch_2 j}(p-q)\}\}$$

【００６４】に設定する等となる。ジョブｑに対してステージ２のプロシージャを使用することで、１とＰの間のプロセッサの各数ｐにおいてジョブ実行時間

【００６５】

【数６】

$$T_q(p) = \hat{t}_1(p)$$

【００６６】が生じる。

【００６７】ステージ３のプロシージャは、サブルーチンとしてプロシージャＳＵＢを用いる。このサブルーチンにおいて、各ジョブｑは固定数$P_q$のプロセッサで実行されるとみなされて、固定された時間の量、例えば$T_q$、で実行する。サブルーチンは、優先順位制約のないようなタスクのスケジュールを見つける。ＳＵＢは、時間の関数として使用されるプロセッサの数のヒストグラムを利用する。当該ヒストグラムは図６の陰影の部分として示される。横軸は使用されるプロセッサの数に対応する。縦軸は時間を表す。タスクが開始及び終了するようスケジュールされるため、ヒストグラムの形状も変化する。ＳＵＢプロシージャの詳細は、図７のブロック図に示されている。最初に、ブロック５０において、タスクはタスク時間の最も長いものから最も短いものに、順に分類される。従って、$T_{q_1} > T_{q_2}$ならばリストの$q_2$の前に$q_1$が現れる。ブロック５２は、順序付けられた実行可能リストの作成を表し、全てのタスクは最初に｛１，．．．Ｑ｝である。時間εの順序付けられたリストもまた維持される。所謂現在の時間τは、常に時間リストの最小の時間である。（時間リストは、実際、現在の時※※間以後の全てのタスク終了時間から成る。）リストは最初に時間τ＝０のみから構成されるため、その時間がプロシージャの開始時における現在の時間である。これらの初期設定は、ブロック５４及び５６に示される。勿論、ブロック５８に示されるように、最初のヒストグラムも同様に０である。現在の時間はＳＵＢの実行中には決して減少しない。更に、タスクはプロシージャが実行するときに、常に現在の時間に等しい開始時間でプロセッサに割り当てられるので、タスクの開始時間は以前に割り当てられたあらゆるタスクの開始時間と少なくとも同じ位の大きさになる。

【００６８】ブロック６０に示されるように、一般的なプロシージャのステップは実行可能リストが空になるまで繰り返される。一般的なステップは以下に示される。図６を考えると、現在の時間τが示されていると仮定する。この時点で使用可能な（即ち、未使用の）プロセッサの数がヒストグラムから読み出される。ＳＵＢは、プロセッサの数と同じ数を必要とするタスクの実行可能リストを検索する。ブロック６２において、そうしたタスクがあるかどうかがテストされる。そうしたタスクがあるならば、リストの最初のタスクｑが選ばれ、ブロック６４でτに等しい開始時間を割り当てて、終了時間がτ＋$T_q$になる。タスクｑはブロック６６でリストから取り除かれ、タイムτ＋$T_q$がブロック６８の時間リストに追加され、ヒストグラムがブロック７０において

【００６９】

【外９】

$$\hat{\tau} - \tau < T_q \text{ となるようにεにおける全ての } \hat{\tau} \text{ で } H(\hat{\tau}) \text{ を } H(\hat{\tau}) + p_q$$

【００７０】に設定することによって更新される。その ようなタスクがないならば、現在の時間τはブロック７

15

2において時間リストから取り除かれ、時間リストの次の最小の時間がブロック７４において新たな現在の時間となる。ヒストグラムは、現在の時間以後は常に単調で増加しない。従って、タスクの開始時間でプロセッサの可用性を検査するのは不十分である。タスクが開始可能ならば、同じプロセッサで干渉なしに終了までを実行することができる。

【００７１】次に、ステージ３自体を示す。図８のブロック図を参照する。各ジョブｑにおいて、全体の作業ｐ。$^{t}$₁（ｐ。）が最小限にされるようにプロセッサの数ｐ。から開始し、Ｔ。＝ｔ。（ｐ。）に設定される。これはブロック８２で示される。ｑループの開始はブロック８０で示され、ループはブロック８４及び８６によって制御される。ブロック８８において、浪費された作業Ｗが非ゼロに設定され、日付(date)を付けることになった最良のメイクスパンＭが無限大に設定される。ブロック９０に示されるように、浪費された作業Ｗが０に駆動されるまで繰り返される。一般的なステップは以下に示される。ブロック９２に示されるように、非柔軟なスケジューリングの問題につきＳＵＢを呼び出す。ブロック９４において、日付を付けることになった最良のメイクスパンＭがＭの以前の値の最小値と、ＳＵＢプロシージャの実行によって見つけられるメイクスパンとに修正される。関数Ｐ－Ｈ（τ）は、各時間τにおける未使用のプロセッサの数を示す。各タスクｑを、ｑが実行していた時間の間に生じた浪費された作業の量と対応することができる。すなわち、

【００７２】

【数７】

$$W_q = \int_{t_q}^{t_q+t_{T_q}} (P - H(\tau))d\tau$$

【００７３】に設定する。この計算はブロック９８で示される。ｑループの開始はブロック９６で示され、ルー＊

16

＊プはブロック１００及び１０２で制御される。最大値Ｗ。を備えたタスクをボトルネックタスクとみなし、ブロック１０４に示されるように浪費された作業Ｗをこの最大値として修正する。そのタスクｑにおいて、プロセッサの数を、最小量の作業ｐＴ。（ｐ）を備えた値ｐ＞ｐ。に増加する。次にブロック１０６に示されるように、ｐ。＝ｐ、Ｔ。＝Ｔ。（ｐ。）に再設定する。ブロック９２において再びＳＵＢを読み出す。プロシージャはボトルネックが０に駆動された固定数のステップで終了する。すなわち、全てのｑでＷ。＝０になる。日付を付けるために得られる最良のメイクスパンが最終的な解決法である。

【００７４】

【発明の効果】本発明は上記より構成され、優先順位を備えたタスクにおいて柔軟なスケジューリングの問題の効率的かつ効果的な解決法が提供される。

【図面の簡単な説明】

【図１】柔軟なスケジューリング問題の入力及び出力を示す高度なフローチャートである。

【図２】本発明に従ったプロシージャの３つのステージの高度なフローチャートである。

【図３】図２のプロシージャの第２ステージのフローチャートである。

【図４】ジョブ優先順位ツリーの一例である。

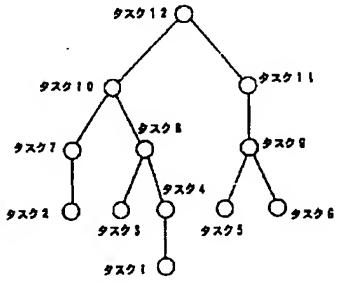
【図５】図４の例に対する第２ステージの解決法の一例である。

【図６】図２のプロシージャの第３ステージにおいてＳＵＢプロシージャによって使用されるヒストグラムである。

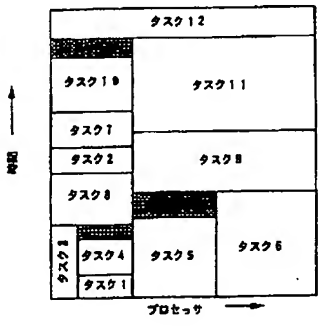
【図７】図２のプロシージャの第３ステージにおけるＳＵＢルーチンのフローチャートである。

【図８】図２のプロシージャの第３ステージのフローチャートである。

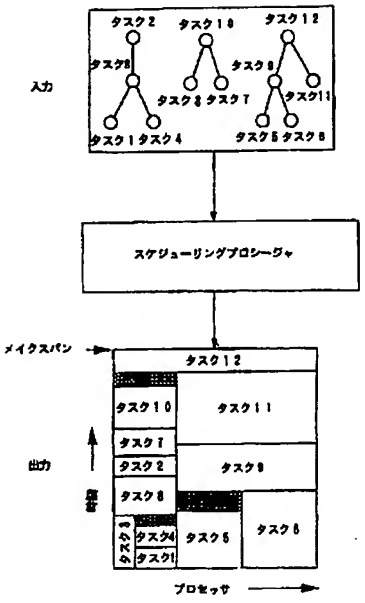
【図４】



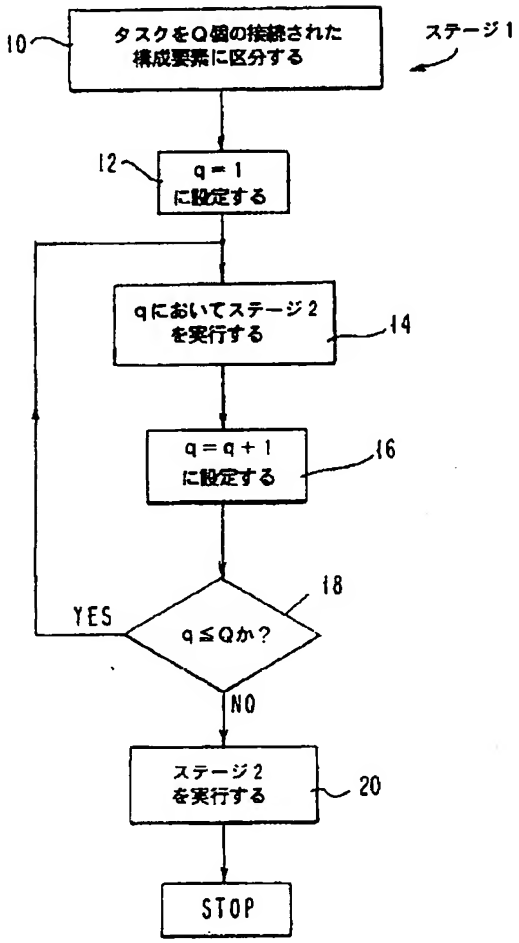

【図５】



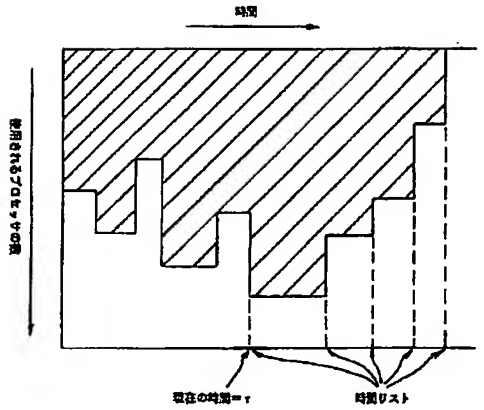
【図1】


タスク2
タスク10
タスク12
タスク9
タスク3
タスク7
タスク11
タスク1
タスク4
タスク5
タスク6
入力
スケジューリングプロシージャ
メイクスパン
タスク12
タスク10
タスク11
タスク7
タスク2
タスク9
タスク3
タスク1
タスク4
タスク5
タスク6
出力
プロセッサ

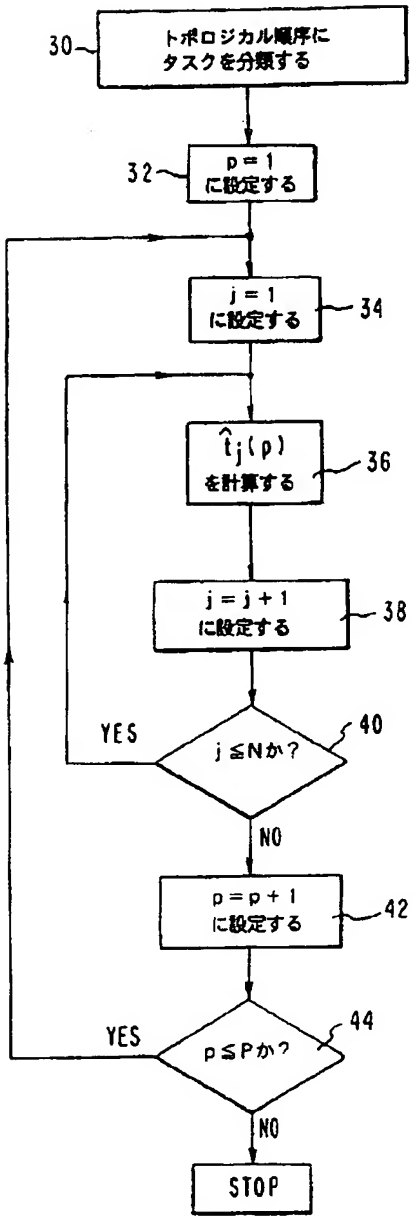
【図2】


10
タスクをQ個の接続された構成要素に区分する
ステージ1
12
q＝1に設定する
qにおいてステージ2を実行する
14
q＝q＋1に設定する
16
18
YES
q≦Qか？
NO
ステージ2を実行する
20
STOP

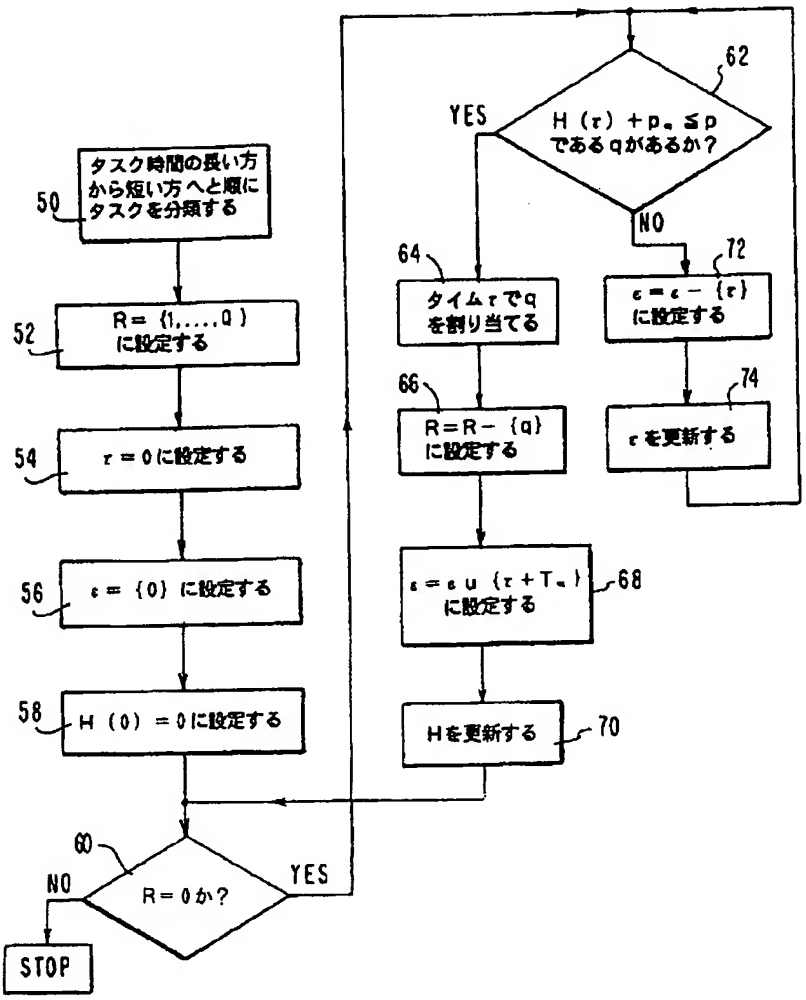
【図6】


時間
現在の時間＝τ
時間リスト

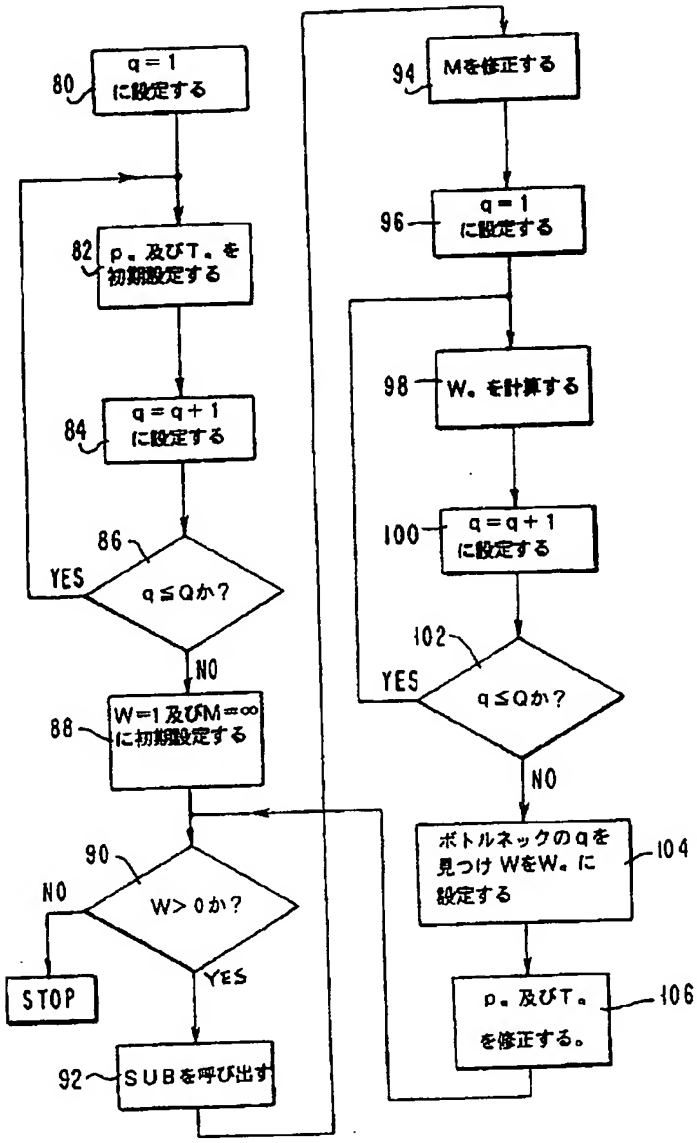
【図3】


30
トポロジカル順序に
タスクを分類する
32
p＝1
に設定する
j＝1
に設定する
34
$\hat{t}_j(p)$
を計算する
36
j＝j＋1
に設定する
38
YES
j≦Nか？
40
NO
p＝p＋1
に設定する
42
YES
p≦Pか？
44
NO
STOP

【図7】


タスク時間の長い方から短い方へと順にタスクを分類する
50
R＝{1,....Q}に設定する
52
r＝0に設定する
54
ε＝{0}に設定する
56
H(0)＝0に設定する
58
R＝0か？
60
NO
YES
STOP
H(r)＋p_q≦pであるqがあるか？
62
YES
NO
タイムrでqを割り当てる
64
R＝R－{q}に設定する
66
ε＝ε∪{r＋T_q}に設定する
68
Hを更新する
70
ε＝ε－{r}に設定する
72
rを更新する
74

【図8】




フロントページの続き

(72)発明者　ジョン　ジョゼフ　エドワード　トュレク
アメリカ合衆国10514、ニューヨーク州チャッパクア、クロス　リッジ　ロード　50

(72)発明者　ジョエル　レオナード　ウォルフ
アメリカ合衆国10536、ニューヨーク州カトナー、チェロキー　コート　7

(72)発明者　フィリップ　シールン　ユー
アメリカ合衆国10514、ニューヨーク州チャッパクア、ストーノワイェ　18